REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	8 August 1996	Final: 1 Apr 93 to 30 April 96

4. TITLE AND SUBTITLE
Multi-Agent Reinforcement Learning and Adaptive Neural Networks

5. FUNDING NUMBERS
F49620-93-1-0269

6. AUTHOR(S)
Andrew G. Barto

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003-4610

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR-96
0487

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM, Building 410
110 Duncan Avenue
Bolling AFB, DC 20332-0001

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unrestricted availability

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This project investigated learning systems consisting of multiple interacting controllers, or agents, each of which employed a modern reinforcement learning method. The objective was to study the utility of reinforcement learning as an approach to complex decentralized control problems. The major accomplishment was a detailed study of multi-agent reinforcement learning applied to a large-scale decentralized stochastic control problem. This study included a very successful demonstration that a multi-agent reinforcement learning system using neural networks could learn high-performance dispatching of multiple elevator cars in a simulated multi-story building. This problem is representative of very large-scale dynamic optimization problems of practical importance that are intractable for standard methods. The performance achieved by the distributed elevator controller surpassed that of the best of the elevator control algorithms accessible in the literature, showing that reinforcement learning can be a useful approach to difficult decentralized control problems. Additional empirical results demonstrated the performance of reinforcement learning systems in the setting of nonzero-sum games, with mixed results. Some progress was also made in improving theoretical understanding of multi-agent reinforcement learning.

14. SUBJECT TERMS	15. NUMBER OF PAGES
stochastic optimal control, learning control, reinforcement learning, multi-agent learning, decentralized control, distributed control	21
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std 239-18
298-102

FINAL TECHNICAL REPORT: AFOSR-F49620-93-1-0269

MULTI-AGENT REINFORCEMENT LEARNING AND ADAPTIVE NEURAL NETWORKS

Principal Investigator: Andrew G. Barto

Abstract–This project investigated learning systems consisting of multiple interacting controllers, or agents, each of which employed a modern reinforcement learning method. The objective was to study the utility of reinforcement learning as an approach to complex decentralized control problems. The major accomplishment was a detailed study of multi-agent reinforcement learning applied to a large-scale decentralized stochastic control problem. This study included a very successful demonstration that a multi-agent reinforcement learning system using neural networks could learn high-performance dispatching of multiple elevator cars in a simulated multi-story building. This problem is representative of very large-scale dynamic optimization problems of practical importance that are intractable for standard methods. The performance achieved by the distributed elevator controller surpassed that of the best of the elevator control algorithms accessible in the literature, showing that reinforcement learning can be a useful approach to difficult decentralized control problems. Additional empirical results demonstrated the performance of reinforcement learning systems in the setting of nonzero-sum games, with mixed results. Some progress was also made in improving theoretical understanding of multi-agent reinforcement learning.

EXECUTIVE SUMMARY

Reinforcement learning (RL) is emerging as a promising method for approximating optimal policies for large-scale stochastic optimal control problems. RL algorithms work by incrementally estimating optimal value functions from samples of control experience, often using artificial neural networks to represent the value-function information. This project investigated learning systems consisting of multiple interacting controllers, or agents, each of which employed an RL method based on the principles of dynamic programming (DP). The objective was to study RL as an approach to complex decentralized control problems.

The major accomplishment was a detailed study of a multi-agent RL system applied to a large-scale decentralized stochastic control problem. This study included a very successful demonstration that a decentralized RL system using neural networks could learn high-performance dispatching of multiple elevator cars in a simulated multi-story building. The performance achieved by the distributed elevator controller surpassed that of the best of the elevator control algorithms accessible in the literature, showing that reinforcement learning can be a useful approach to difficult distributed control problems. Although not of obvious direct relevance to the Air Force mission, the elevator dispatching problem was selected because: 1) it is representative of a class of very large-scale stochastic dynamic optimization problems that have practical significance; 2) it is intractable for standard solution methods, such as stochastic DP (it was formulated as a semi-Markov decision process

with 10^{22} states, which is decisively beyond the capabilities of conventional DP); and 3) it is naturally formulated as a distributed problem: each elevator had its own controller. Other instances of such problems are relevant to control strategies for reducing computational overhead, to distributed communication strategies, and to other problems relevant to cost-containment in large-scale operations. All of these problems have in common a high degree of computational complexity that rules out exact solution methods.

A simulator for a 10-floor/4-elevator system was used. The control actions of the elevator cars were executed asynchronously in continuous time. The performance objective was to minimize the sum of the squared wait times of all waiting passengers. Table compares the average waiting time, the average squared waiting time, and the average system time (sum of waiting and travel time) averaged over 30 simulated hours of down-peak operation, for a number of controllers. The two bottom rows give the performances of two RL controllers (RLd and RLp) that we developed. The remaining rows give performances of a number of other controllers, including controllers that are the best of which we are aware from the literature. Significant performance improvement was achieved by the RL controllers using policies learned through approximately 60-thousand hours of simulated operation, taking several days on a 100 MIPS workstation. Although this is a considerable amount of computation, it is negligible compared to what any conventional DP algorithm would require (thousands of years).

Table 1: *Comparative Elevator Dispatching Results for Down-Peak Traffic.* (AvgWait = Average Wait Time, SqWait = Squared Wait Time, and SysTime = System Time)

Algorithm	AvgWait	SqWait	SysTime
SECTOR	21.4	674	47.7
DLB	19.4	658	53.2
BASIC HUFF	19.9	580	47.2
LQF	19.1	534	46.6
HUFF	16.8	396	48.6
FIM	16.0	359	47.9
ESA/nq	15.8	358	47.7
ESA	15.1	338	47.1
RLd	14.9	326	41.8
RLp	14.4	296	42.1

The details of this study will be available in the Ph.D. dissertation of R. H. Crites, one of the computer science graduate students supported by this grant. At the time of writing this report, however, this dissertation was not yet complete (although several short conference papers had appeared). A technical summary of this work is provided below.

Most RL research has been confined to single agent settings or to multi-agent settings where the agents receive identical rewards (team problems like the elevator dispatching problem). RL has also been studied in the setting of zero-sum games [12]. However, very little

research exists on RL applied to nonzero-sum games, which are substantially more difficult than the other cases. An empirical study was conducted of RL applied to the nonzero-sum game known as the iterated prisoner's dilemma (IPD). We investigated the ability of a variety of RL agents to play the IPD game against an unknown opponent. In some experiments, the opponent was the fixed strategy Tit-for-Tat, while in others it was another RL agent. All the RL agents learned to play optimally against Tit-for-Tat. Playing against other learning agents was more difficult due to their nonstationary, and because they were not given an *a priori* policy designed to encourage cooperation. The learning agents that were studied varied along three dimensions: the length of history they received as context, the type of memory they employed (lookup tables based on restricted history windows versus recurrent neural networks that can theoretically store features from arbitrarily deep in the past), and the exploration schedule they followed. Although all the learning agents faced difficulties when playing against other learning agents, agents with longer history windows, lookup-table memories, and longer exploration schedules fared best in the IPD games. Details of this research are provided below and in refs. [15; 16].

Although the results obtained with the elevator dispatching problem provide evidence that multi-agent RL can be very successful in practice, multi-agent RL is not well understood theoretically. Researchers studying learning automata [12] have developed a number of convergence results for single-stage multi-agent RL tasks, where the objective for a team of agents is to learn a mapping from situations to actions that maximizes their expected *immediate* rewards. We worked toward extending this theoretical framework to the more difficult case of sequential tasks, where the objective is to maximize the expected *long-term* reward. We developed a multi-agent policy improvement algorithm which we conjectured is guaranteed to cause the agents to monotonically improve their expected infinite-horizon discounted reward from every state. We were unable to prove this conjecture by the end of the period being reported but are continuing this line of investigation.

In work funded by a previous AFOSR grant (AFOSR-89-0526) we developed an RL algorithm applicable to problems with continous action spaces (most RL algorithms apply to discrete–action problems). A neuron–like unit, called a Stochastic Real–Valued (SRV) unit, embodies this algorithm and has been used in artificial neural networks that learn by reinforcement feedback. We developed a network using SRV units to learn how to insert a peg into a hole using a Zebra Zero robot arm. This approach was very successful, resulting in a system that learned how to perform insertions in the champherless case with very small clearances despite large uncertainties introduced by this rather imprecise robot arm. Work under the present grant successfully extended this system to the three-dimensional case and to a square-peg/square-hole case. Details of this work are contained in ref. [8].

In summary, the results obtained using the elevator dispatching testbed provide evidence that multi-agent RL can be useful in large-scale decentralized stochastic control problems. In fact, because the elevator dispatching problem is intractable for standard methods, this study is being cited as a major example showing the utility of RL, whether multi-agent or not. Problems in which the agents do not receive identical rewards, however, are more difficult for RL as illustrated by our study of the iterated prisoner's dilemma game. Although RL agents always learned optimal policies against fixed opponents, they did not perform

consistenty against other learning agents. These results show that more sophisticated RL algorithms are required for multi-agent learning in the setting of nonzero-sum games and suggest features that more successful algorithms should have.

Publications Resulting from This Project:

Barto, A. G., Reinforcement Learning and Dynamic Programming. *Proceedings of the 6th IFAC/IFIP/FORS/IEA Symposium on Analysis, Design and Evaluation of Man-Machine Systems* (MMS'95), 27-29 June 1995, Cambridge, MA, pp. 469–474.

Crites, R. H. (1995) Reinforcement Learning with Compact Function Approximation: A Case Study from the Elevator Dispatching Domain. In: Boyan, J. A., Moore, A. W., and Sutton, R. S., eds., *Proceedings of the Workshop on Value Function Approximation, Machine Learning Conference 1995.* Carnegie Mellon University Technical Report CMU-CS-95-206. Available online at: http://www.cs.cmu.edu/ reinf/ml95/

Crites, R. H. and Barto, A. G. (1995) Improving Elevator Performance Using Reinforcement Learning. In: Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., eds., *Advances in Neural Information Processing Systems 8.* MIT Press, Cambridge, MA, pp. 1017-1023. Available online at: ftp://ftp.cs.umass.edu/pub/anw/pub/crites/nips8.ps.Z

Crites. R. H. and Barto, A. G., Forming Control Policies from Simulation Models using Reinforcement Learning. *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems,* New Haven, CT, June 10-12, 1996.

V. Gullapalli and A. G. Barto. "Convergence of Indirect Asynchronous Indirect Adaptive Asynchronous Value Iteration Algorithms" *Neural Information Processing Systems 6* (NIPS-6), J.D. Cown, G. Tesauro and J. Alspector, Eds. San Mateo: Morgan Kaufmann, 1994. pp. 695-702.

V. Gullapalli, R. Grupen, and A. G. Barto. "Learning Admittance Mappings for Force–Guided Assembly" *Proc. of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, May 1994.

V. Gullapalli, J. Franklin, and H. Benbrahim. "Acquiring Robot Skills via Reinforcement Learning" *IEEE Control Systems Magazine*, Feb. 1994.

Sandholm, T. W. and Crites, R. H. (1995) Multiagent Reinforcement Learning in the Iterated Prisoner's Dilemma. *Biosystems* 37:147-166, Special Issue on the Prisoner's Dilemma. Available online at: ftp://ftp.cs.umass.edu/pub/lesser/sandholm-biosystems95.ps

Sandholm, T. W. and Crites, R. H. (1995) On Multiagent Q-learning in a Semi-Competitive Domain. *Proceedings of the Workshop on Adaptation and Learning in Multiagent Systems,* 14th International Joint Conference on Artificial Intelligence (IJCAI-95) Montreal, Canada, pp. 71-77.

Sandholm, T. W. and Crites, R. H. (1996) On Multiagent Q-learning in a Semi-Competitive Domain. In: Weiss, G. and Sen, S., eds., *Adaptation and Learning in Multiagent Systems*. Lecture Notes in Artificial Intelligence, Volume 1042. Springer-Verlag, Berlin.

Conference Presentations:

Barto, A. G., Reinforcement Learning and Dynamic Programming. Presented at the IFAC Conference on Man-Machine Systems (MMS'95), 27-29 June 1995, Cambridge, MA.

Barto, A. G., Forming Control Policies from Simulation Models using Reinforcement Learning. Presented at the Ninth Yale Workshop on Adaptive and Learning Systems, New Haven, CT, June 10-12, 1996.

Crites, R.H., Reinforcement Learning with Compact Function Approximation: A Case Study from the Elevator Dispatching Domain. Presented at the Workshop on Value Function Approximation, Machine Learning Conference, Tahoe City, CA. July 9, 1995.

Crites, R.H., Improving Elevator Performance Using Reinforcement Learning. At NIPS-95 Conference (Advances in Neural Information Processing Systems), Denver, CO. November 29, 1995.

Crites, R.H., An Actor/Critic Algorithm that is Equivalent to Q-Learning. At NIPS-94 Conference (Advances in Neural Information Processing Systems), Denver, CO. November 29. 1994. (Poster presentation)

Research Interactions:

Joint research with T. Sandholm, an RA with Professor Lesser in Computer Science. Also with Christos Cassandras, Professor of Electrical and Computer Engineering.

Personnel Supported:

Andrew G. Barto, Professor
Robert Crites, Graduate Research Assistant
Sergio Guzman-Lara, Graduate Research Assistant
Leonid Kuvayev, Graduate Research Assistant
Gwyn Mitchell, Secretary (part-time support)
Janet Turnbull, Grants Manager (part-time support).

1 Distributed Elevator Dispatching

The problem of efficiently dispatching elevators in multi-story buildings is an example of a stochastic optimal control problem that is well beyond the capabilities of classical stochastic DP due to the curse-of-dimensionality. It is relevant to the Air Force mission because it is representative of a class of large-scale scheduling and resource allocation problems that can have economic significance in large organizations. Elevator dispatching is a familiar problem to anyone who has used an elevator system, but in spite of its conceptual simplicity, it poses significant difficulties. Elevator systems operate in high-dimensional continuous state spaces and in continuous time as discrete-event dynamic systems. Their states are not fully observable and they are nonstationary due to changing passenger arrival rates. Because an optimal policy for elevator group control is not known, existing control algorithms were used as standards for comparison. The elevator problem also provided an opportunity to develop and compare several contol architectures, and to monitor the amount of degradation that occurs as the controllers faced decreasing levels of state information.

Elevator systems are driven by passenger arrivals. Arrival patterns vary during the course of the day. In a typical office building, the morning rush hour brings a peak level of up traffic, while a peak in down traffic occurs during the afternoon. Other parts of the day have their own characteristic patterns. Different arrival patterns have very different effects, and each pattern requires its own analysis. (Up-peak and down-peak elevator traffic are not simply equivalent patterns in opposite directions.) Up-peak traffic is the easiest to analyze because all passengers enter cars at the lobby, their destination floors are serviced in ascending order, and empty cars then return to the lobby. The optimal policy for handling pure up-peak traffic is thought to involve closing the doors of a car stopped at the lobby after an optimal number of passengers have entered the car or an optimal amount of time has elapsed [18]. Two-way and down-peak traffic patterns require many more decisions in a wider variety of contexts than does pure up traffic. For this reason, a down-peak traffic pattern was used as a testbed for our investigations of distributed RL in this domain. Before describing the testbed in detail, the various approaches to elevator control are briefly discussed.

1.1 Elevator Control Strategies

The oldest relay-based automatic controllers used the principle of *collective control* [18; 17], where cars always stop at the nearest call in their running direction. One drawback of this scheme is that there is no means to avoid the phenomenon called *bunching*, where several cars arrive at a floor at about the same time, making the interval, and thus the average waiting time, much longer. Advances in electronics, including the advent of microprocessors, made possible more sophisticated control policies.

Zoning Approaches—The Otis Elevator Company has used zoning as a starting point in dealing with various traffic patterns [18]. Each car is assigned a zone of the building. It answers hall calls within its zone, and parks there when it is idle. The goal of zoning is to keep the cars reasonably well separated and thus keep the interval down. The drawback of

this approach seems to be a significant loss of flexibility [2; 3]. Sakai & Kurosawa [14] of Hitachi describe a related approach called *area control*.

Search-Based Approaches—Another strategy is to search through the space of all possible car assignments, selecting the one that optimizes some criterion such as the average waiting time. Tobita et al. [19] of Hitachi describe a system using a fuzzy rule-based system to pick the coefficients and estimating functions by which wait time, travel time, and riding ratio are predicted. Receding horizon controllers are also examples of search-based approaches. After every event, they perform an expensive search for the best assignment of hall calls assuming no new passenger arrivals. Closed-loop control is achieved by re-calculating a new open-loop control sequence after every event. The weaknesses of this approach are its computational demands and its lack of consideration of future arrivals. Examples of receding horizon controllers are Finite Intervisit Minimization (FIM) and Empty the System Algorithm (ESA) [2], which we used for comparative purposes. FIM attempts to minimize squared waiting times and ESA attempts to minimize the length of the current busy period, the same objective as Levy et al [9].

Rule-Based Approaches—Ujihara and Tsuji [20] of Mitsubishi describe a system that uses expert-system and fuzzy-logic technologies. They claim that experts in group-supervisory control have the experience and knowledge necessary to shorten waiting times under various traffic conditions, but admit that expert knowledge is fragmentary, hard to organize, and difficult to incorporate.

Other Heuristic Approaches—The Longest Queue First (LQF) algorithm assigns upward moving cars to the longest waiting queue, and the Highest Unanswered Floor First (HUFF) algorithm assigns upward moving cars to the highest queue with people waiting [2]. Both of these algorithms are designed specifically for down-peak traffic. They assign downward moving cars to any unassigned hall calls they encounter. The Dynamic Load Balancing (DLB) algorithm attempts to keep the cars evenly spaced by assigning contiguous non-overlapping sectors to each car in a way that balances their loads [10]. We also use these heuristic algorithms for comparative purposes.

Adaptive and Learning Approaches—A variety of adaptive and learning approaches have been proposed for the elevator dispatching problem. The most relevant of these is by Levy et al. [9], who use dynamic programming (DP) offline to minimize the expected time needed for completion of the current busy period. The trouble with using DP to calculate an optimal policy is that the state space is very large, requiring drastic simplification. Levy et al. use several methods to keep the size of the state space manageable: they consider a building with only 2 cars and 8 floors, where the number of buttons that can be on simultaneously is restricted, the state of the buttons are restricted to binary values (i.e., elapsed times are discarded), and the cars have unlimited capacity. Zoning is also mentioned as a way to reduce the number of states in higher buildings. Construction of the transition probability matrix is the principle part of the procedure, and it assumes that the intensity of Poisson arrivals at each floor is known. Value iteration or policy iteration is then performed to obtain the solution. However, it seems doubtful that their minimization procedure could be effective if the number of cars were increased from 2 to 4, or if the elapsed times of the hall buttons were taken into account.

1.2 The Elevator Testbed

The particular elevator system we examined was a simulated 10-story building with 4 elevator cars. The simulator was a discrete-event simulator written by Lewis [10]. Passenger arrivals at each floor were assumed to be Poisson, with arrival rates that vary during the course of the day. Our simulations used a traffic profile [2] that dictated arrival rates for every 5-minute interval during a typical afternoon down-peak rush hour. Table 1 shows the mean number of passengers arriving at each floor (2-10) during each 5-minute interval who were headed for the lobby. In addition, there was inter-floor traffic which varied from 0% to 10% of the traffic to the lobby.

Table 2: *The down–peak traffic profile.*

Time	00	05	10	15	20	25	30	35	40	45	50	55
Rate	1	2	4	4	18	12	8	7	18	5	3	2

The system dynamics were approximated using the following parameters:

- Floor time (the time to move one floor at the maximum speed): 1.45 secs.
- Stop time (the time needed to decelerate, open and close the doors, and accelerate again): 7.19 secs.
- Turn time (the time needed for a stopped car to change direction): 1 sec.
- Load time (the time for one passenger to enter or exit a car): random variable from a $20th$ order truncated Erlang distribution with a range from 0.6 to 6.0 secs and a mean of 1 sec.
- Car capacity: 20 passengers.

Although the simulator was quite detailed, a few minor deviations from reality should be noted. In the simulator, a car could accelerate to full speed or decelerate from full speed in a distance of only one half of a floor, while the distances would be somewhat longer in a real system. Thus, the simulated acceleration and deceleration times were always the same, whereas in a real system they vary depending on the speed of the elevator. For example, an express car descending from the tenth floor at top speed takes longer to decelerate at the first floor than a car that is descending from the second floor. The simulator also allowed the cars to commit to stopping at a floor when they are only one half of a floor away. Although this is not realistic for cars moving at top speed, the concept remains valid of making decisions regarding the next floor where a car can commit to stopping.

State Space—The state space was continuous because it included the elapsed times since any hall calls were registered, which were real-valued. Even if these real values were approximated as binary values, the size of the state space would still be immense. Its components

include 2^{18} possible combinations of the 18 hall call buttons (up and down buttons at each landing except the top and bottom), 2^{40} possible combinations of the 40 car buttons, and 18^4 possible combinations of the positions and directions of the cars (rounding off to the nearest floor). Other parts of the state were not fully observable, for example, the exact number of passengers waiting at each floor, their exact arrival times, and their desired destinations. Ignoring everything except the configuration of the hall and car call buttons and the approximate position and direction of the cars, we obtain an extremely conservative estimate of the size of a discrete approximation to the continuous state space:

$$2^{18} \cdot 2^{40} \cdot 18^4 \approx 10^{22} \text{ states.}$$

Performing any of the classical DP algorithms on a problem of this size is effectively impossible.

Control Actions—Each car had a small set of primitive control actions. If it was stopped at a floor, it must either "move up" or "move down". If it was in motion between floors, it must either "stop at the next floor" or "continue past the next floor". Due to passenger expectations, there were two constraints on these actions: a car could not pass a floor if a passenger wanted to get off there and could not turn until it had serviced all the car buttons in its present direction. We added three additional action constraints in an attempt to build in some primitive prior knowledge: a car could not stop at a floor unless someone wanted to get on or off there, it could not stop to pick up passengers at a floor if another car was already stopped there, and given a choice between moving up and down, it should prefer to move up (since the down-peak traffic tended to push the cars toward the bottom of the building). Because of this last constraint, the only choices left to each car were the stop and continue actions. The actions of the elevator cars were executed asynchronously since they could take different amounts of time to complete.

Performance Criteria—One possible objective is to minimize the average *wait* time, which is the time between the arrival of a passenger and his entry into a car. Another possible objective is to minimize the average *system* time, which is the sum of the wait time and the travel time. A third possible objective is to minimize the percentage of passengers that wait longer than some dissatisfaction threshold (usually 60 seconds). Another common objective is to minimize the sum of *squared* wait times. We focused on this latter performance objective since it tended to keep the wait times low while also encouraging fair service.

1.3 The Algorithm and Network Architecture

Here we describe the multi-agent reinforcement learning algorithm that was applied to elevator group control. In our scheme, each agent was responsible for controlling one elevator car. Each agent learned via a modification of Q-learning for discrete-event systems. Together, they employed a collective form of reinforcement learning. We begin by describing the modifications needed to extend Q-learning to a discrete-event framework, and derive a method for determining appropriate reinforcement signals in the face of uncertainty about exact passenger arrival times. Then we describe the algorithm, the feedforward networks

used to store the Q-values, and the distinction between parallel and distributed versions of the algorithm.

Discrete-Event Reinforcement Learning—Elevator systems can be modeled as discrete event systems [5], where significant events (such as passenger arrivals) occur at discrete times, but the amount of time between events is a real-valued variable. In such systems, the constant discount factor γ used in most discrete-time reinforcement learning algorithms is inadequate. This problem can be approached using a variable discount factor that depends on the amount of time between events [4]. In this case, returns are defined as integrals rather than as infinite sums, as follows:

$$\sum_{t=0}^{\infty} \gamma^t r_t \quad \text{becomes} \quad \int_0^{\infty} e^{-\beta\tau} r_\tau d\tau,$$

where r_t is the immediate cost at discrete time t, r_τ is the instantaneous cost at continuous time τ (e.g., the sum of the squared wait times of all waiting passengers), and β controls the rate of exponential decay.

Because elevator system events occur randomly in continuous time, the branching factor is effectively infinite, which complicates the use of algorithms that require explicit lookahead. Therefore, we employed a discrete event version of the Q-learning algorithm since it considers only events encountered in actual system trajectories and does not require an explicit model of the state transition probabilities. The Q-learning update rule [21] takes on the following discrete event form [4]:

$$\Delta\hat{Q}(x,a) = \alpha \cdot [\int_{t_x}^{t_y} e^{-\beta(\tau - t_x)} r_\tau d\tau + e^{-\beta(t_y - t_x)} \min_b \hat{Q}(y,b) - \hat{Q}(x,a)],$$

where action a is taken from state x at time t_x, the next decision is required from state y at time t_y, α is the learning rate parameter, and r_τ and β are defined as above. $e^{-\beta(t_y - t_x)}$ acts as a variable discount factor that depends on the amount of time between events. Unless otherwise noted, $\beta = 0.01$ in the simulations described below.

Bradtke and Duff [4] consider the case where r_τ is constant between events. We extended their formulation to the case where r_τ is quadratic, since our goal was to minimize squared wait times. The integral in the Q-learning update rule then takes the form:

$$\int_0^{t_y - t_x} \sum_p e^{-\beta\tau} (\tau + w_p)^2 d\tau,$$

where w_p is the amount of time each passenger p has already waited at time t_x.

A difficulty arises in calculating this integral since it requires knowledge of the waiting times of all waiting passengers. However, only the waiting times of passengers who press hall call buttons will be known in a real elevator system. The number of subsequent passengers to arrive and their exact waiting times will not be available. We examined two ways of

dealing with this problem. The simulator has access to the waiting times of all passengers. We can use this information to produce the necessary reinforcement signals. We called this *omniscient* reinforcement since it requires information that would not be available in a real system. Omniscient reinforcement can be used during the off-line design phase of an elevator controller. Once the controller is installed in a real system, however, any fine tuning would have to be done without the benefit of this extra knowledge.

The other possibility is to use only information that would be available to a real system on-line. Such *on-line* reinforcement assumes only that the waiting time of the first passenger in each queue is known (which is the elapsed button time). If the Poisson arrival rate λ for each queue is known or can be estimated, the Gamma distribution can be used to estimate the arrival times of subsequent passengers. The time until the n^{th} subsequent arrival follows the Gamma distribution $\Gamma(n, \frac{1}{\lambda})$. For each queue, subsequent arrivals will generate the following expected penalties during the first b seconds after the hall button has been pressed:

$$\sum_{n=1}^{\infty} \int_0^b (\text{prob } n^{th} \text{ arrival occurs at time } \tau) \cdot (\text{penalty given arrival at time } \tau)\, d\tau$$

$$= \sum_{n=1}^{\infty} \int_0^b \frac{\lambda^n \tau^{n-1} e^{-\lambda\tau}}{(n-1)!} \int_0^{b-\tau} w^2 e^{-\beta(w+\tau)} dw\, d\tau \;=\; \int_0^b \int_0^{b-\tau} \lambda w^2 e^{-\beta(w+\tau)} dw\, d\tau.$$

This integral can be solved by parts to yield expected penalties. Using on-line reinforcement produces results that are almost as good as those obtained with omniscient reinforcement.

Collective Discrete-Event Q-Learning—Elevator system events can be divided into two types. Events of the first type are important in calculating waiting times and therefore also reinforcements. These include passenger arrivals and transfers in and out of cars in the omniscient case, or hall button events in the online case. The second type are car arrival events, which are potential decision points for the RL agents controlling each car. A car that is in motion between floors generates a car arrival event when it reaches the point where it must decide whether to stop at the next floor or continue past the next floor. In some cases, cars are constrained to take a particular action, for example, stopping at the next floor if a passenger wants to get off there. An agent faces a decision point only when it has an unconstrained choice of actions.

The algorithm used by each agent is as follows:

1. At time t_x, observing state x, car c arrives at a decision point. It selects an action using the Boltzmann distribution over its Q-value estimates:

$$Pr(stop) = \frac{e^{Q(x,cont)/T}}{e^{Q(x,stop)/T} + e^{Q(x,cont)/T}},$$

where T is a positive "temperature" parameter that is decreased during training. Call the action selected action a.

2. Let the next decision point for car c be at time t_y in state y. It adjusts its estimate of $Q(x, a)$ toward the following target value:

$$\int_{t_x}^{t_y} e^{-\beta(\tau - t_x)} r_\tau d\tau + e^{-\beta(t_y - t_x)} \min_b \hat{Q}(y, b),$$

where r_τ and β are defined as above.

3. Let $x \leftarrow y$ and $t_x \leftarrow t_y$. Go to step 1.

The Networks Used to Store the Q-Values—Using lookup tables to store the Q-values was ruled out for such a large system. Instead, we used feedforward neural networks trained with the error backpropagation algorithm [13]. The networks receive some of the state information as input, and produce Q-value estimates as output. Some of our experiments used separate single-output networks for each action–value estimate, while others used one network with multiple output units. Our basic network architecture for pure down traffic has 47 input units, 20 hidden sigmoid units, and 1 or 2 linear output units. The input units are as follows:

- 18 units: Two units encode information about each of the nine down hall buttons. A real–valued unit encodes the elapsed time if the button has been pushed and a binary unit is on if the button has not been pushed.
- 16 units: Each of these units represents a possible location and direction for the car whose decision is required. Exactly one of these units will be on at any given time.
- 10 units: These units each represent one of the 10 floors where the other cars may be located. Each car has a "footprint" that depends on its direction and speed. For example, a stopped car causes activation only on the unit corresponding to its current floor, but a moving car causes activation on several units corresponding to the floors it is approaching, with the highest activations on the closest floors.
- 1 unit: This unit is on if the car whose decision is required is at the highest floor with a waiting passenger.
- 1 unit: This unit is on if the car whose decision is required is at the floor with the passenger that has been waiting for the longest amount of time.
- 1 unit: The bias unit is always on.

Parallel and Distributed Implementations—Each elevator car is controlled by a separate Q-learning agent. We experimented with several different control architectures. All were decentralized in the sense that the control decisions were made separately for each elevator car, although using information about the states of the other cars. The *parallel* architecture consisted of a single two–output network, or a single pair of one–output networks, that learned from the experiences of all the cars and that were consulted by all the cars for

decision making. This architecture forces the cars to learn identical policies. What we call the *decentralized* architecture, on the other hand, consisted of four two–output networks, or four pairs of one–output networks, one for each elevator car. The network (or network pair) for each car learned only from the experience of that car and was consulted for decision making by that car. This architecture allowed each controller to learn a specialized policy. In either case, none of the controllers had explicit access to the actions of the other controllers. Cooperation had to be learned indirectly via the global reinforcement signal. All of the control architectures produced policies that would be simple to implement and execute in real-time on an actual elevator installation, although this step has not yet been taken.

1.4 Main Results

Since an optimal policy for the elevator dispatching problem is unknown, we measured the performance of our algorithm against other algorithms, including the best of which we were aware. The algorithms were: SECTOR, a sector–based algorithm similar to what is used in many actual elevator systems; DLB, Dynamic Load Balancing, attempts to equalize the load of all cars; HUFF, Highest Unanswered Floor First, gives priority to the highest floor with people waiting; LQF, Longest Queue First, gives priority to the queue with the person who has been waiting for the longest amount of time; FIM, Finite Intervisit Minimization, a receding horizon controller that searches the space of admissible car assignments to minimize a load function; ESA, Empty the System Algorithm, a receding horizon controller that searches for the fastest way to "empty the system" assuming no new passenger arrivals. FIM is very computationally intensive, and would be difficult to implement in real time in its present form. ESA uses queue length information that would not be available in a real elevator system. ESA/nq is a version of ESA that uses arrival rate information to estimate the queue lengths. For more details, see [2].

RLp and RLd denote the RL controllers, parallel and decentralized. In each case, single–output networks were used. The RL controllers were each trained on 60,000 hours of simulated elevator time, which took four days on a 100 MIPS workstation. The results are averaged over 30 hours of simulated elevator time. Table (above) shows the results for the traffic profile with down traffic only. Table 3 shows the results for the down–peak traffic profile with up and down traffic, including an average of 2 up passengers per minute at the lobby. The algorithm was trained on down–only traffic, yet it generalizes well when up traffic is added and upward moving cars are forced to stop for any upward hall calls. Table 4 shows the results for the down–peak traffic profile with up and down traffic, including an average of 4 up passengers per minute at the lobby. This time there is twice as much up traffic, and the RL agents generalize extremely well to this new situation.

One can see that both RL systems achieved very good performance, most notably as measured by system time (the sum of the wait and travel time), a measure that was not directly being minimized. Surprisingly, the decentralized RL system was able to achieve nearly as good a level of performance as the parallel RL system. We also note that these results, which were produced using separate networks for each action, are better than those obtained for two–output networks [6]. For example, in the case of down traffic only, the best

Table 3: *Results for Down-Peak Profile with Up and Down Traffic.*

Algorithm	AvgWait	SqWait	SysTime	%>60s
SECTOR	27.3	1252	54.8	9.24
DLB	21.7	826	54.4	4.74
BASIC HUFF	22.0	756	51.1	3.46
LQF	21.9	732	50.7	2.87
HUFF	19.6	608	50.5	1.99
ESA	18.0	524	50.0	1.56
FIM	17.9	476	48.9	0.50
RLd	16.9	465	42.5	1.29
RLp	16.8	454	42.8	1.20

Table 4: *Results for Down-Peak Profile with Twice as much Up Traffic.*

Algorithm	AvgWait	SqWait	SysTime	%>60s
SECTOR	30.3	1643	59.5	13.50
HUFF	22.8	884	55.3	5.10
DLB	22.6	880	55.8	5.18
LQF	23.5	877	53.5	4.92
BASIC HUFF	23.2	875	54.7	4.94
FIM	20.8	685	53.4	3.10
ESA	20.1	667	52.3	3.12
RLd	19.0	614	45.8	2.71
RLp	18.7	583	45.6	2.29

average waiting time, average squared waiting time, and system time achieved by the parallel architecture using two–output networks are respectively 14.8, 320, and 41.8 (compare with the last line of Table).

In view of the outstanding success of the decentralized RL algorithm on this problem, we conducted a large number of additional experiments designed to help us better understand the nature of the decentralized results. Results of these experiments will be availabe in Crites' dissertation.

1.5 Summary of Elevator Dispatching Studies

Although we do not know how the results described above compare to optimal elevator dispatching performance (because no optimal policy is known), they show that RL is capable of producing high–performance control of this large-scale stochastic system. Moreover, they provide evidence that RL can be useful in certain large-scale distributed control problems. The chief difficulty with problems of this kind lies not in the creation of an accurate simulation model, but rather in using conventional stochastic DP solution methods, which are intractable for problems of this complexity due to the large number of states. By using a simulation model to produce a large number of sample trajectories, RL avoids exhaustive sweeps of the state set. Computation is successively focused onto more and more relevant subsets of states because the control policy being learned is used to generate control decisions during the simulation. Additionally, by storing information using parameterized function approximation methods, here, artificial neural networks, RL avoids the need to maintain large lookup tables. Finally, unlike conventional stochastic DP, this approach does not require access to an explicit representation of the probability structure of the controlled system. The simulation model is simply a source of sample experience. It is often much easier to construct a simulation model than it is to provide an explicit probabilistic description of the system.

2 Multi-agent RL in the Iterated Prisoner's Dilemma

Most RL research has been confined to single agent settings or to multi-agent settings where the agents have totally positively correlated payoffs (team problems) or totally negatively correlated payoffs (zero-sum games). We conducted an empirical study of RL in the iterated prisoner's dilemma (IPD), where the agents' payoffs are neither totally positively nor totally negatively correlated. RL is considerably more difficult in such a domain. We investigated the ability of a variety of Q-learning agents to play the IPD game against an unknown opponent. In some experiments, the opponent was the fixed strategy Tit-for-Tat, while in others it was another Q-learning agent. The learning agents that were studied varied along three dimensions: the length of history they received as context, the type of memory they employed (lookup tables based on restricted history windows versus recurrent neural networks that can theoretically store features from arbitrarily deep in the past), and the exploration schedule they followed.

2.1 Prisoner's Dilemma

The *prisoner's dilemma* (PD) game is an abstraction of social situations where each agent is faced with two alternative actions: *cooperating*, i.e., doing the socially responsible thing, and *defecting*, i.e., acting according to self-interest regardless of how harmful this might be to other agents. We concentrated on the 2-agent PD game, where the agents' payoffs for different action combinations are common knowledge. Characteristically, each agent is better off defecting regardless of the opponent's choice, but the sum of the agents' payoffs is maximized if both agents choose to cooperate—thus the dilemma. In game theoretic terms, defecting is a dominant strategy of the game and so the defect-defect action combination is the only *dominant strategy equilibrium* (and therefore also the only *Nash equilibrium*). On the other hand, *social welfare* is maximized at the cooperate-cooperate action combination, if social welfare is defined to be the equiweighted sum of the agents' payoffs. The payoff matrix of Table 5 defines a PD game if the following inequalities hold:

$$T > R > P > S, \quad 2R > T + S > 2P.$$

The PD game is *noncooperative*: no pregame negotiation is allowed, the agents cannot bindingly commit to any action, no enforced threats can be made, and no transfer of payoff is possible.

		column player	
		cooperate (C)	defect (D)
row player	cooperate (C)	R = 0.3, R = 0.3	S = 0.0, T = 0.5
	defect (D)	T = 0.5, S = 0.0	P = 0.1, P = 0.0

Table 5: Payoff Matrix with Row Player's Payoffs Shown First. The numerical values are the ones used in the experiments. In general, T, R, P, and S could differ between the agents as long as they define a PD game.

In practical situations, agents often encounter each other more than once. Correspondingly, some social interactions can be modeled by a sequence of PD games. This *supergame* of the PD game is called the *iterated prisoner's dilemma* (IPD) game. In supergames, an agent's policy (strategy) is a mapping from the entire history (all of its own and its opponent's moves) to an action. Because fixed horizon IPD games have this characteristic, we focused on IPD games with an indefinite horizon, i.e., the agents do not know how many iterations are still to come. In such IPD games, it may be beneficial even for a selfish agent to cooperate on some iterations in the hope of invoking cooperation from its opponent. The goal of an agent at iteration n is to select actions that will maximize its infinite-horizon discounted return.

Describing an intelligent strategy for a supergame is difficult because arbitrarily long input histories must be considered. Strategy designers have used two main approaches to address this problem:

- use only a fixed number of previous moves as the context upon which the choice of next action is based, and

- iteratively keep a tally of some (numeric) features that provide an abstract characterization of the entire history.

An example of the first approach is the pure strategy Tit-for-Tat (TFT), which cooperates on the first move and then does whatever the opponent did on the previous move. An example of the second approach is to compute at each time step the opponent's discounted cumulative score. A strategy has the desired property of being *collectively stable* if and only if it defects when that score exceeds a threshold [1]. Both approaches to the problem of growing context suffer from the *hidden state problem*: the first approach ignores the older history, and the second approach can only give an abstraction of the true state.

There is no single best strategy for the IPD game. Which strategy is best depends on the opponent's strategy, which the player obviously does not know. TFT was chosen as an opponent for the RL players not only because it has performed well in IPD tournaments and evolutionary IPD experiments [1], but also because the optimal way to play against TFT is completely known. There are three different optimal ways to play against TFT depending on the discount factor γ [1]. For the payoff matrix of the PD game we used ($T = 0.5, R = 0.3, P = 0.1, S = 0.0$), optimal play is achieved by always cooperating if $\gamma \geq \frac{2}{3}$, alternating between defection and cooperation if $\frac{1}{4} \leq \gamma < \frac{2}{3}$, and always defecting if $\gamma < \frac{1}{4}$. Each of the three ways of playing can be achieved by a number of strategies. For example, cooperation with TFT is realized by another TFT strategy or by a strategy that always cooperates no matter what the opponent does.

2.2 Q-Learning Players In The IPD

We generated a number of different types of Q-learning agents to challenge TFT and each other in IPD games. In each pairing of players in each experiment, the whole learning session was one long trial (300,000 PD iterations), i.e. the agent had only one chance to learn and evaluation took place during the last 100 iterations. The results were averaged over 100 IPD games for each pairing. Unless otherwise stated, the experiments were run with discount factor $\gamma = 0.95$ (high to promote cooperation). The agents differed in the way they stored Q-values.

The first set of RL agents used lookup tables to store their Q-values. These agents had access only to w previous moves—i.e., w of their own moves and w of their opponent's moves. For example, given a window of only the last move ($w = 1$), four different inputs would be possible (CC, CD. DC, and DD). For each possible input, two Q-values needed to be stored (corresponding to actions C and D).

The second set of agents had the same inputs, but they stored the Q-values in a recurrent neural network (RNN) that can (at least in theory) store information of arbitrarily old actions and automatically learn which history features are important. Inputs were presented to the net in four bits because a unary encoding resulted in faster learning than a two bit binary encoding. The first bit was on if the opponent's previous action was C, and the second bit was on if the action was C. The third bit was on if the agent's own previous action was C,

and the fourth bit was on if its action was C. We used a separate net for both actions, which has been shown empirically to enhance learning speed [11]. Each net was constructed along the lines of Elman [7]: the net was a normal feedforward net except that the hidden-unit outputs were copied into context units, whose activations were fed back to the normal hidden units on the next forward sweep. The copy connections from the normal hidden units to the context units were fixed to one. This allowed the use of the standard backpropagation learning algorithm.

In our experiments, each learning agent's exploration policy was a function of the number n of PD games played so far. This was used to decrease the temperature for Boltzmann exploration:

$$t = 5 \cdot 0.999^n.$$

If $t < 0.01$ then no exploration was performed, i.e. the certainty equivalence action (the action with the highest Q-value estimate) was always chosen. The constants for the annealing schedule $(5, 0.999, 0.01)$ were chosen experimentally. 5 and 0.01 are specific to the range of Q-values defined by the payoff matrix. The parameter α referred to below is the step-size parameter of the Q-learning algorithm.

2.3 Experiments

Learning Against TFT—This experiment was designed to see how the different learning agents performed against a player that used a fixed strategy (TFT). Lookup-table Q-learning agents using only the last move as the input consistently learned to cooperate with TFT for a variety of parameter settings. RNN-based Q-learners also learned to cooperate with TFT, as did a learner that had the same neural-net architecture as the RNN but with no context units and no feedback connections. Lookup-table learners learned to cooperate with TFT in thousands of iterations, while RNN players required tens of thousands of iterations.

We also examined whether a lookup-table Q-learner could learn the optimal strategy against TFT for values of γ other than 0.95. At each setting of γ, we ran 100 IPD with 100,000 iterations each. For $\gamma = 0.05$, $\gamma = 0.1$, $\gamma = 0.15$, and $\gamma = 0.2$, the agent consistently learned to defect. For $\gamma = 0.25$, $\gamma = 0.3$,..., $\gamma = 0.65$, the agent learned to alternate between defecting and cooperating. For $\gamma = 0.7$, $\gamma = 0.75$,..., $\gamma = 0.95$, the agent learned to cooperate. Thus, the agent learned to play optimally against TFT in all of the hundred IPD games at each setting of γ.

Q-learning, though relatively slow, works extremely well against stationary policies such as TFT, which take into account a short window of previous moves. Playing against an agent with a stationary policy is analogous to single agent learning, because the learning agent perceives a stationary environment. The next experiments addressed harder cases, where both agents learn simultaneously.

Simultaneous Learning—Two types of learning agents were studied in the context of simultaneous learning: lookup-table Q-learners and RNN Q-learners. All three pairings of these agents were tested. When two lookup-table players played each other, one never totally took

advantage of the other, but asymmetric loops did occur (e.g. CC, CD, CC, CD, ...). Between RNN players, neither total advantage taking nor asymmetric loops occurred. When lookup-table learners played against RNN learners, surprisingly asymmetric loops only occurred to the *advantage of the lookup table player.*

The sum of the agents' payoffs was much higher in the game between lookup-table learners than in the other cases. If even one of the players used a RNN as its Q-value storage, the outcome of the game was significantly less cooperative. This somewhat surprising result may be because the neural-net players need more training examples than the lookup table players and thus may require a longer exploration phase than lookup-table players.

We also explored the impact of turning off the exploration. The results of games where at least one agent does no exploration depend heavily on the initialization of the Q-values. For example, if all Q-values are initialized with the same non-positive number, a non-exploring agent will always pick the action that it first picked in that state. This is because the Q-value corresponding to that action will be reinforced by a positive number (and thus exceed any other action's Q-value in that state), because the payoffs in the game are positive. In the experiments, the Q-values were initialized randomly from a uniform distribution from 0 to 1.These results suggest that exploration is crucial to avoid being taken advantage of by an exploring opponent. Yet if neither agent explores, the Q-value initializations determine the outcome, and cooperation occurs frequently.

We also investigated what happens when learners having different inputs play each other. The joint behavior of the agents was very sensitive to the annealing schedule (rates 0.999 and 0.9999 tested). Slower annealing tended to produce significantly more cooperation and other semi-cooperative loops and a wider variety of final looping patterns, while the faster annealing schedule increased defection. Agents with history windows of 1, 2, and 3 moves were tested. With longer histories, a wider variety of patterns developed, and longer patterns developed. In the asymmetric contests, the agent with the longer history tended to fare slightly better than the agent with the shorter history, but not as much as had been expected. Overall, there was clearly more cooperation when both agents used a history of one move.

2.4 Summary of Iterated Prisoner's Dilemma Results

Both lookup-table Q-learners and recurrent neural-net Q-learners learned to play optimally against a fixed policy (TFT) for all settings of the discount factor. Having multiple agents learning simultaneously without communication, however, makes the learning process considerably more difficult due to the nonstationarity that each agent faces. In our experiments with two learning systems, clear cooperation seldom emerged, although the discount factor was set very high to stimulate cooperation. Recurrent-net learners played non-cooperatively, and the resulting plays were always equal among the players. Lookup-table learners played more cooperatively, but asymmetric loops occurred. When a recurrent neural-net learner played a lookup-table learner, the outcomes were non-cooperative, and the asymmetric loops were always to the lookup-table learner's advantage. Surprisingly, increasing the learning rate α from 0.2 to 1 enhanced cooperation between recurrent net

players. Lookup-table learners with memories of different lengths were tested. With longer histories, a wider range of interaction patterns occurred, and longer loops developed. With asymmetric memories, the agent with a longer memory did slightly better on average. With asymmetric and long symmetric memories, the outcomes were less cooperative than when both agents had a memory of length one.

References

[1] R. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, NY, 1984.

[2] G. Bao, C. G. Cassandras, T. E. Djaferis, A. D. Gandhi, and D. P. Looze. Elevator dispatchers for down peak traffic. ECE Department technical report, University of Massachusetts, 1994.

[3] S. M. Benmakhlouf and S. K. Khator. Smart lifts: Control design and performance evaluation. *Computers and Industrial Engineering*, 25:175–178, 1993.

[4] S. J. Bradtke and M. O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing 7*, pages 393–400. Cambridge, MA, 1995. MIT Press.

[5] C. G. Cassandras. Discrete Event Systems: Modeling and Performance Analysis. Irwin Publishers, Burr Ridge, IL, 1993.

[6] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1017–1023, Cambridge, MA, 1996. MIT Press.

[7] J. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[8] V. Gullapalli, A. G. Barto, and R. A. Grupen. Learning admittance mappings for force-guided assembly. In *Proceedings of the 1994 International Conference on Robotics and Automation*, pages 2633–2638, 1994.

[9] D. Levy, M. Yadin, and A. Alexandrovitz. Optimal control of elevators. *International Journal of Systems Science*, 8:301–320, 1977.

[10] J. Lewis. *A Dynamic Load Balancing Approach to the Control of Multiserver Polling Systems with Applications to Elevator System Dispatching.* PhD thesis, University of Massachusetts, September 1991.

[11] L. J. Lin. *Reinforcement Learning for Robots using Neural Networks.* PhD thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1993. Technical Report CMU-CS-93-103.

[12] K. Narendra and M. A. L. Thathachar. *Learning Automata: An Introduction.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations.* Bradford Books/MIT Press, Cambridge, MA, 1986.

[14] Y. Sakai and K. Kurosawa. Development of elevator supervisory group control system with artificial intelligence. *Hitachi Review*, 33:25–30, 1984.

[15] T. W. Sandholm and R. H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37:147-166, 1995.

[16] T. W. Sandholm and R. H. Crites. On multiagent Q-learning in a semi-competitive domain. 14th International Joint Conference on Artificial Intelligence (IJCAI-95) Workshop on Adaptation and Learning in Multiagent Systems. Montreal, Canada, pp. 71-77, 1995.

[17] Marja-Liisa Siikonen. Elevator traffic simulation. *Simulation*, 61:257–267, 1993.

[18] G. R. Strakosch. *Vertical Transportation: Elevators and Escalators.* Wiley and Sons, New York, 1983.

[19] T. Tobita, A. Fujino, H. Inaba, K. Yoneda, and T. Ueshima. An elevator characterized group supervisory control system. In *Proceedings of IECON*, pages 1972–1976. 1991.

[20] H. Ujihara and S. Tsuji. The revolutionary ai-2100 elevator-group control system and the new intelligent option series. *Mitsubishi Electric Advance*, 45:5–8, 1988.

[21] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.